THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Efficient Encrypted Keyword Search for Multi-user Data Sharing

Citation for published version:
Kiayias, A, Oksuz, O, Russell, A, Tang, Q & Wang, B 2016, Efficient Encrypted Keyword Search for Multi-user Data Sharing. in I Askoxylakis, S Ioannidis, S Katsikas & C Meadows (eds), *Computer Security -- ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I.* Lecture Notes in Computer Science (LNCS), vol. 9878, Springer International Publishing, Cham, pp. 173-195, 21st European Symposium on Research in Computer Security, Crete, Greece, 26/09/16. https://doi.org/10.1007/978-3-319-45744-4_9

Digital Object Identifier (DOI):
10.1007/978-3-319-45744-4_9

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Computer Security -- ESORICS 2016

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. May. 2023

Efficient Encrypted Keyword Search for Multi-User Data Sharing

Aggelos Kiayias[1], Ozgur Oksuz[2], Alexander Russell[2], Qiang Tang[3], and Bing Wang[2]

[1] University of Edinburgh, UK
[2] University of Connecticut, USA
[3] Cornell University/NJIT
{aggelos,acr}@cse.uconn.edu
{ozgur.oksuz,bing}@engr.uconn.edu
qt44@cornell.edu

Abstract. In this paper, we provide a secure and efficient encrypted keyword search scheme for multi-user data sharing. Specifically, a data owner outsources a set of encrypted files to an untrusted server, shares it with a set of users, and a user is allowed to search keywords in a subset of files that he is authorized to access. In the proposed scheme, (a) each user has a constant size secret key, (b) each user generates a constant size trapdoor for a keyword without getting any help from any party (e.g., data owner), independent of the number of files that he is authorized to search, and (c) for the keyword ciphertexts of a file, the network bandwidth usage (from the data owner to the server) and storage overhead at the server do not depend on the number of users that are authorized to access the file. We show that our scheme has data privacy and trapdoor privacy. While several recent studies are on secure keyword search for data sharing, we show that they either suffer from scalability issues or lack user privacy.

Key words: data sharing, keyword search, broadcast encryption

1 Introduction

Cloud computing has become a prevalent and economic platform for users to outsource and share data. For instance, through a file or picture sharing service (e.g., Dropbox, Flickr), users can conveniently upload files or pictures to the cloud to share them with their friends, family or colleagues. In a health information sharing system, a number of research institutes and hospitals may share medical data to facilitate collaboration and accelerate scientific discovery. In such data sharing applications, the sensitive nature of the data means that it is necessary for data owners to encrypt the data before outsourcing it to the cloud. On the other hand, useful functions, e.g., searching over the outsourced encrypted data, should still be supported to preserve the utility of the data.

Another complication in such data sharing applications is that data owners often selectively share data with others. For instance, Alice may want to share family pictures with her family members, while share work related files with her colleagues. As such, a user can only have access to a subset of the files that are permitted by the corresponding data owners. Since different users have access to different files, a natural way for encryption is that a data owner encrypts different files using different keys. After that, a data owner gives each user the encryption/decryption keys of all the files that the user is authorized to search. This trivial solution is clearly inefficient. First, the number of keys that the data owner needs to send to a user depends on the number of files. Secondly, the number of trapdoors (or tokens) that a user needs to submit when searching for a keyword also depends on the number of files.

In a recent study [25], Popa and Zeldovich solve the above problem by proposing a *multi-key searchable encryption* scheme that allows a user to provide a *single* search token to the server, while allowing the server to search for the (encrypted) keyword in documents encrypted with *different* keys. Their solution requires each data owner to send the server some public information (delta values) for each user that is allowed to search a file. In general, suppose a data owner has m files, each file is shared with up to n users, and each file contains up to k keywords, the data owner needs to outsource $O(m(n+k))$ amount of data to the server to support keyword search, where $O(mn)$ and $O(mk)$ correspond to delta values and keyword ciphertexts, respectively. As a result, both the network bandwidth overhead (from the data owner to the server) and storage overhead (at the server) are $O(m(n+k))$. While the overhead associated with outsourcing keyword ciphertexts is unavoidable, it is desirable to reduce the overhead for outsourcing the delta values, which can dominate the overhead when $k \ll n$ (e.g., for a picture with a small number of keywords but shared with many users).

Our contributions. Our contributions are as follows:

- We give an efficient encrypted keyword search scheme for multi-user data sharing that overcome the drawbacks in [25]. Specifically, in our scheme, when a data owner has m files, each file has up to k keywords, to support encrypted keyword search by up to n users, the data owner only needs to outsource $O(mk)$ amount of information to the cloud. Given that outsourcing keyword ciphertexts is unavoidable (which is $O(mk)$), our scheme is asymptotically optimal in network bandwidth usage and cloud storage. In addition, each user has constant size secret keys and generates constant size trapdoor for a keyword search in all the files that he is authorized to search by a data owner.
- Our scheme requires very different techniques to compress ciphertexts and succinctly represent the search/access policies that dictate which files a user can access. We model and analyze the security of our construction and provide detailed proofs for keyword, file and trapdoor privacy, respectively.
- We give an extension that allows a system with multiple trusted parties (e.g., managers, data owners) to generate distributively the public key for

the system and secret keys for the users, thus eliminating the need of having a single trusted party.

Related Work. To the best of our knowledge, no existing study achieves the efficiency as ours in the setting that we study. Our work is related to searchable encryption in general. Single-user searchable encryption considers a single data owner and only the data owner is allowed to submit search queries. It has been studied extensively, with focus on symmetric-key settings [13, 16, 19, 27], asymmetric-key settings [5, 8], supporting complicated (conjunctive, subset, or boolean) queries [10, 12], or dynamic settings [21]. Our work is in multi-user setting where a group of users can submit search queries.

Multi-user searchable encryption. While multi-user searchable encryption has been investigated in a number of studies, these studies differ from our study in important aspects. In [16], a user is allowed to search *all* the files owned by a data owner while in our study each user is allowed to search a different *subset* of the files. The study in [4] relies on a fully trusted user manager, which not only sets system public parameters but also generates secret keys of all parties. In addition, when a data owner outsources his data, he needs to interact with the semi-trusted server, and the keyword ciphertext size depends on the number of users. Similar to [4], the study in [17] also uses a fully trusted third party. In addition, their scheme does not provide an access control mechanism that specifies which users can access which files. A recent study [20] proposes multi-user searchable encryption that supports boolean queries. It does not provide an access control mechanism either. In addition, an authorized user needs to interact with the data owner to get a token. Then, he can generate a trapdoor based on the token for the query. The same limitations hold for [18], which extends [20] to support range, substring, wildcard and phrase queries.

In the above studies, the documents are encrypted with a single key. In other studies, the documents are encrypted with different keys. We have mentioned [25], which has motivated our study. Tang [28] improves the study of [25] by presenting a new security model for multiparty searchable encryption and proposing a scheme that authorizes users to share their files. Liu et al. [24] propose a data sharing scheme, which however has two major drawbacks. First, a user needs to interact with a honest-but-curious server to download some information in advance to generate a trapdoor. Secondly, the trapdoor size is linear in the number of files. Cui et al. [15] adapt the idea in [14] and propose a key-aggregate searchable encryption scheme. In their scheme, a data owner provides a single aggregated key that contains information about all files that a user is authorized to search. Each user is given a secret key (i.e., the aggregated key) and then a user generates constant size trapdoor to search a keyword from all files. Their scheme is, however, vulnerable to cross pairing attack, and hence has neither data privacy nor trapdoor privacy. Basically, an untrusted server can figure out the encrypted keyword from the ciphertext and recover the aggregate key of any user (see more details in Appendix A.1). Rompay et al. [29] propose a scheme that uses a proxy based architecture to achieve a stronger security model. Their scheme suffers from the same inefficiency problem as that in [25].

Other related primitives for encrypted keyword search. The study [8] introduces a scheme that transforms a given identity based encryption (IBE) to public key encryption with keyword search (PEKS). A later study [1] examines the relationships between these two primitives and points out that if one has an anonymous IBE scheme, then he achieves secure PEKS. Attrapadung et al. [2, 3] introduce a new cryptographic primitive called *coupling*. It is a broadcast encryption that further has hierarchical identity based dimension. It combines user index with identity. They provide restricted anonymous identity based encryption. Their scheme does not have a concrete proof of security. Moreover, it cannot be applied to our setting since the transformation does not provide trapdoor privacy. Last, existing studies [23, 30] investigate attribute based keyword search, which differ significantly from our study. In [30], the access control policy is based on keywords instead of files, while [23] does not allow a secret key holder to generate search token (trapdoor) individually. In [23], the trapdoor is generated by the collaboration between a secret key holder and the fully trusted PKG (private key generator). In addition, in both schemes, the keyword cliphertext, and trapdoor and secret key sizes depend on the total number of attributes that are involved in a data owner's access control policy.

2 Preliminaries

Bilinear Map: (1) $\mathbf{G}_1$, $\mathbf{G}_2$ and $\mathbf{G}'$ are three multiplicative cyclic groups of prime order p; (2) g_1 is a generator of $\mathbf{G}_1$, g_2 is a generator of $\mathbf{G}_2$. A bilinear map $e : \mathbf{G}_1 \times \mathbf{G}_2 \rightarrow \mathbf{G}'$ has the following properties: (1) for all $u \in \mathbf{G}_1, v \in \mathbf{G}_2$ and $a, b \in Z_p$, we have $e\left(u^a, v^b\right) = e\left(u, v\right)^{ab}$; (2) the map is not degenerate, i.e., $e\left(g_1, g_2\right) \neq 1$. It is a symmetric bilinear map when $\mathbf{G}_1 = \mathbf{G}_2 = \mathbf{G}$.

2.1 Popa-Zeldovich Scheme [25]

We briefly summarize the construction in [25]. For simplicity, we only present the scenario where a single data owner has m files (each file is encrypted with a different key) and can search all the files with a single trapdoor (token).

Let $H : \{0,1\}^* \rightarrow \mathbf{G}_1$, $H_2 : \mathbf{G}' \times \mathbf{G}' \rightarrow \{0,1\}^*$ be hash functions that are modeled as random oracles. Let $e : \mathbf{G}_1 \times \mathbf{G}_2 \rightarrow \mathbf{G}'$ be a bilinear map. The multi-key searchable encryption (MKSE) scheme proposed in [25] is as follows.

- **MK.Setup(λ):** return params $(p, \mathbf{G}_1, \mathbf{G}_2, \mathbf{G}, e, g_1, g_2, g')$.
- **MK.KeyGen(params):** returns $uk, k_1, ..., k_m$, where uk is the user secret key, k_j is the encryption key of file j.
- **MK.Delta($uk, k_1, ..., k_m$):** returns $\Delta_j = g_2^{k_j/uk} \in \mathbf{G}_2$.
- **MK.Token(uk, w):** returns $tk_w = H(w)^{uk} \in \mathbf{G}_1$.
- **MK.Enc(k_j, w):** Draw $r \in \mathbf{G}'$, outputs $c_{j,w} = (r, h)$, where $h = H_2(r, e(H(w), g_2)^{k_j})$.
- **MK.Adjust(tk_w, Δ_j):** returns $stk_{j,w} = e(tk_w, \Delta_j) = e(H(w)^{uk}, g_2^{k_j/uk})$.
- **MK.Match($stk_{j,w}, c_{j,w}$):** parse $c_{j,w}$ as (r, h) and check if $H_2(r, stk_{j,w}) \stackrel{?}{=} h$. if so, outputs $b = 1$, otherwise $b = 0$.

In the above construction, to search T different keywords, a user only needs to provide $O(T + m)$ pieces of information to the server (T trapdoors and m delta values), much more efficient than that when using standard searchable encryption (which needs $O(Tm)$ pieces of information).

The authors show that the MKSE scheme has data hiding and token hiding properties. Data hiding (privacy) requires that the semi-honest adversary is not able to distinguish between ciphertexts of two values not matched by some token. Token hiding (privacy) requires that the adversary cannot learn the keyword that one searches for.

The MKSE scheme is inefficient since the data owner needs to provide a delta value for each user that is authorized to search a file. Hence the number of delta values for a file is linear in the number of users. This results a significant communication overhead between a data owner and the server when the data owner outsources the delta values, and significant storage overhead for the server to store these ciphertexts.

2.2 Complexity Assumptions

Decision Linear Assumption. Let $\mathbf{G}$ be a bilinear group of prime order p. The Decision Linear problem [7] in $\mathbf{G}$ is stated as follows: given a vector $(g, Z_1 = g^{z_1}, Z_2 = g^{z_2}, Z_{13} = g^{z_1 z_3}, Z_{24} = g^{z_2 z_4}, Z) \in \mathbf{G}^6$ as input, determine whether $Z = g^{z_3+z_4}$ or a random value $R \in \mathbf{G}$. The advantage of an algorithm $\mathcal{B}$ in deciding the decision linear problem in $\mathbf{G}$ is

$$\left|\Pr\left[\mathcal{B}\left(g, g^{z_1}, g^{z_2}, g^{z_1 z_3}, g^{z_2 z_4}, g^{z_3+z_4}\right) = 0\right] - \Pr\left[\mathcal{B}\left(g, g^{z_1}, g^{z_2}, g^{z_1 z_3}, g^{z_2 z_4}, R\right) = 0\right]\right| \leq \epsilon\,.$$

ℓ-Bilinear Diffie-Hellman Exponentiation Assumption (ℓ-BDHE) [6]. Let $\mathbf{G}$ be a bilinear group of prime order p. The ℓ-BDHE problem in $\mathbf{G}$ is defined as follows: given a vector of $2\ell + 1$ elements $\left(h, g, g^{\alpha}, g^{\alpha^2}, ..., g^{\alpha^{\ell}}, g^{\alpha^{\ell+2}}, ..., g^{\alpha^{2\ell}}\right)$ as input, output $e\left(g, h\right)^{\alpha^{\ell+1}}$. Once g and α are specified, we denote $g_i = g^{\alpha^i}$ as shorthand. An algorithm $\mathcal{A}$ has advantage ϵ in solving ℓ-BDHE in $\mathbf{G}$ if

$$\Pr\left[\mathcal{A}\left(h, g, g^{\alpha}, ..., g^{\alpha^{\ell}}, g^{\alpha^{\ell+2}}, ..., g^{\alpha^{2\ell}}\right) = e\left(g^{\alpha^{\ell+1}}, h\right)\right] \geq \epsilon,$$

where the probability is over the random choice of generators $g, h \in \mathbf{G}$, the random choice of $\alpha \in Z_p$ and random bits used by $\mathcal{B}$. The decisional version of the ℓ-BDHE problem in $\mathbf{G}$ is defined analogously. Let $\bar{\boldsymbol{y}}_{g,\alpha,\ell} = \left(g^{\alpha}, \ldots, g^{\alpha^{\ell}}, g^{\alpha^{\ell+2}}, \ldots, g^{\alpha^{2\ell}}\right)$. An algorithm $\mathcal{B}$ that outputs $b \in \{0, 1\}$ has advantage ϵ in solving decision ℓ-BDHE in $\mathbf{G}$ if

$$\left|\Pr\left[\mathcal{B}\left(g, h, \bar{\boldsymbol{y}}_{g,\alpha,\ell}, e\left(g_{\ell+1}, h\right)\right) = 0\right] - \Pr\left[\mathcal{B}\left(g, h, \bar{\boldsymbol{y}}_{g,\alpha,\ell}, R\right) = 0\right]\right| \geq \epsilon,$$

where the probability is over the random choice of generators g, h in $\mathbf{G}$, the random choice of $\alpha \in Z_p$, the random choice of $R \in \mathbf{G}'$, and the random bits consumed by $\mathcal{B}$.

n-Decisional Diffie-Hellman Inverse Assumption (n-DDHI). Let $\mathbf{G}$ be a bilinear group of prime order p. The n-DDHI assumption in $\mathbf{G}$ is stated as follows: given a vector $g, \bar{\boldsymbol{y}}_{g,\alpha,n} = (g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}), g^{\gamma}, h = g^{z}, Z \in \mathbf{G}^{2n+3}$ as input, determine whether $Z = h^{\frac{\gamma}{\alpha^{n+1}}}$ or a random value R in $\mathbf{G}$. The advantage of an algorithm $\mathcal{B}$ in deciding the n-DDHI in $\mathbf{G}$ is

$$\left|\Pr\left[\mathcal{B}\left(g, \bar{\boldsymbol{y}}_{g,\alpha,n}, g^{\gamma}, g^{z}, g^{\frac{z\gamma}{\alpha^{n+1}}}\right) = 0\right] - \Pr\left[\mathcal{B}\left(g, \bar{\boldsymbol{y}}_{g,\alpha,n}, g^{\gamma}, g^{z}, R\right) = 0\right]\right| \leq \epsilon.$$

We provide security evidence of our hardness assumption (n-DDHI) by presenting bounds on the success probabilities of an adversary $\mathcal{A}$. We follow the theoretical generic group model (GGM) as presented in [26] and show the justification of it.

Theorem 1. *Let $\mathcal{A}$ be an algorithm that solves the n-DDHI problem in the generic group model, making a total of at most q queries to the oracles computing the group action in $\mathbf{G}, \mathbf{G}'$ and the oracle computing the bilinear pairing e. If $\alpha, \gamma, r, z \in Z_p^*$ and ξ, ξ' are chosen at random, then*

$$\Pr\left[\begin{array}{c}\mathcal{A}(p, \xi(1), \xi(z), \xi(\alpha), \ldots, \xi(\alpha^n), \\ \xi(\alpha^{n+2}), .., \xi(\alpha^{2n}), \xi(\gamma), \xi(t_0), \xi(t_1)) = d : \\ \alpha, \gamma, z \leftarrow Z_p^*, d \leftarrow \{0,1\}, t_d \leftarrow \frac{z\gamma}{\alpha^{n+1}}, t_{1-d} \leftarrow r\end{array}\right] \leq \frac{1}{2} + \frac{16n(q+n+1)^2}{p}$$

We provide security evidence for the hardness of the n-DDHI in Appendix A.2.

3 Secure and Efficient Multi-user Encrypted Keyword Search (SEMEKS)

In this section, we define the notion of secure and efficient multi-user encrypted keyword search (SEMEKS) for data sharing. The constructions are deferred to Section 4.

Definition 1. *In SEMEKS, there are n users, m documents (files) and a server. Let U denote the set of users and $D = \{M_1, \ldots, M_m\}$ denote the set of documents. Each document has a set of unique keywords. Let F_j denote the set of unique keywords in M_j. Let $S_j \subseteq U$ denote the set of users that can access file j. If user $i \in S_j$, then he is able to search any keyword in F_j and retrieve M_j.*

*The server $\mathcal{S}$ stores all ciphertexts for keywords and documents. The server is **honest-but-curious**, i.e., he does not change any data, and he gives the query results honestly, but he is curious in that he is trying to learn more information from the data and queries (for extracting keywords from ciphertexts and trapdoors). Our model does not allow the adversary (the server) to collude with any of the users. Otherwise, it leaks keywords.*

Definition 2. *A SEMEKS scheme consists of the following five algorithms.*

- ***Setup***(n, λ)*: a randomized algorithm that takes the number of users* n*, and the security parameter* λ*, as input. It outputs* $(pk, \{sk_1, \ldots, sk_n\})$*, where* pk *is the set of system public key and* sk_i *is the secret key of user* i*.*
- ***Enc***(pk, F_k, S_k, M_k)*: a randomized algorithm that takes file* M_k*, the set of unique keywords* F_k*, public key* pk*, and user set* S_k *as input. It outputs keyword ciphertext* C_k *and file ciphertext* C_k''*.*
- ***Trap***(sk_i, w)*: a randomized algorithm that takes secret key* sk_i *and keyword* w *as input. It outputs trapdoor* $t_{i,w}$ *for user* i *to query* w*.*
- ***Test***$(pk, S_k, t_{i,w}, C_k)$*: a deterministic algorithm that takes keyword ciphertext* C_k*, trapdoor* $t_{i,w}$*, user set* S_k*, and public key* pk *as input. It outputs* $b \in \{0, 1\}$*, where* $b = 1$ *if* $i \in S_k$ *and* C_k *includes the ciphertext of* w*; otherwise,* $b = 0$*.*
- ***Dec***(pk, sk_i, C_k'', S_k)*: a deterministic algorithm. If* $b = 1$*, then the algorithm takes public key* pk*, secret key* sk_i*, file ciphertext* C_k'' *and user set* S_k *as input, and outputs the plaintext file* M_k*. Otherwise, it outputs* $\perp$*.*

We consider keyword privacy, file privacy, and trapdoor privacy. In addition, we refer to keyword and file privacy together as data privacy. The security definitions of keyword and trapdoor privacy are similar as those defined in [25] with some differences (our encryption scheme uses asymmetric key, and hence the adversary is able to encrypt keywords himself in our setting, which differs from that in [25]). File privacy is defined in our study while not in [25] since it does not consider file decryption functionality.

We define a semantically secure keyword privacy game. In the game, $\mathcal{A}$ is static that he outputs a keyword and a set pair that he wants to be challenged on. He observes encryption of keywords and trapdoors. However, he is not able to distinguish whether the challenge ciphertext is encoded by the challenge keyword or a random keyword.

Definition 3 (Keyword Privacy). *We define static semantic security for keyword privacy in SEMKS by the following game between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{C}$*. Both* $\mathcal{C}$ *and* $\mathcal{A}$ *are given* (n, λ) *as input.*

- ***Init:*** $\mathcal{A}$ *takes security parameter* λ *and outputs a set* S_0*, a keyword* w^* *that he wants to be challenged on.*
- ***Setup:*** $\mathcal{C}$ *runs Setup*(n, λ) *algorithm to obtain system public key* pk *and a set of private keys,* $sk_1, ..., sk_n$*. It then gives the public key* pk *to* $\mathcal{A}$*.*
- ***Query:*** $\mathcal{A}$ *adaptively issues queries* $q_1, ..., q_\lambda$*, where query* q_k *is a trapdoor query* (i, w)*. For such a query,* $\mathcal{C}$ *responds by running algorithm Trap*(sk_i, w) *to derive* $t_{i,w}$*, and sends it to* $\mathcal{A}$*.*
- ***Guess:*** $\mathcal{C}$ *picks a random number* $b \in \{0, 1\}$*, computes* $(C_0, S_0) \leftarrow Enc(pk, w_b, S_0)$*, where* $w_0 = w^*$*, and* w_1 *is a random keyword (of the same length as* w_0*), returns the value* (C_0, S_0) *to* $\mathcal{A}$*.* $\mathcal{A}$ *outputs its guess* $b' \in \{0, 1\}$ *for* b *and wins the game if* $b = b'$*.*

Restriction*: The adversary asks trapdoor queries only when* $i \notin S_0$ *and* $w \neq w^*$*.*

A SEMKS scheme is keyword private if, for all PPT adversaries $\mathcal{A}$, for all sufficiently large λ, $\Pr[\text{win}_{\mathcal{A}}(\lambda, n)] < 1/2 + \text{negl}(\lambda, n)$, where $\text{win}_{\mathcal{A}}(\lambda, n)$ is a random variable indicating whether the adversary wins the game for security parameter λ.

We define a file privacy game that is similar to a semantically secure broadcast encryption definition since a file is encrypted for a set of users. If a user is a member of the corresponding set, he can download the file ciphertext and retrieves the file using his decryption key. In file privacy game, $\mathcal{A}$ outputs two messages M_0, M_1 and a user set S_0 he wants to be challenged upon. He gets public parameters and user secret keys that are not in the target user set (S_0). However, he is not able to distinguish whether the given ciphertext is the ciphertext of M_0 or M_1.

Definition 4 (File Privacy). *We define static semantic security for file privacy in SEMEKS by the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.*

- ***Init:*** *$\mathcal{A}$ takes parameters (n, λ) and outputs a set S_0 and two messages M_0, M_1 that he wants to be challenged on.*
- ***Setup:*** *$\mathcal{C}$ runs Setup(n, λ) algorithm to obtain system public key pk. It then gives the public key pk to $\mathcal{A}$.*
- ***Query:*** *$\mathcal{A}$ adaptively issues private key queries of user $j \notin S_0$. $\mathcal{C}$ gives the secret keys, sk_j to $\mathcal{A}$, where $j \notin S_0$.*
- ***Challenge:*** *$\mathcal{C}$ chooses a random $b \in \{0, 1\}$ and runs $Enc(S_0, pk, M_b)$ to obtain the ciphertext C''^*, and gives it to $\mathcal{A}$.*
- ***Guess:*** *$\mathcal{A}$ guesses $b' \in \{0, 1\}$ for b and wins the game if $b = b'$.*

File privacy game is secure against CPA if for all attacks $\left|\Pr[b = b'] - \frac{1}{2}\right| \leq \text{negl}(\lambda, n)$.

We define a static trapdoor privacy game that $\mathcal{A}$ outputs challenge user index and keyword pair at the beginning of the game to be challenged on. In the game, $\mathcal{A}$ can observe encryptions of keywords and the challenge trapdoor, but he is not able to distinguish the challenge keyword from a random keyword.

Definition 5 (Trapdoor Privacy). *The trapdoor privacy game is between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ as follows:*

- ***Init:*** *$\mathcal{A}$ takes parameters (n, λ) and outputs a user and keyword tuple (i^*, w^*) that he wants to be challenged on.*
- ***Setup:*** *$\mathcal{C}$ runs Setup(n, λ) to obtain system public key pk and private keys $sk_1, ..., sk_n$. It gives pk to $\mathcal{A}$.*
- ***Query:*** *$\mathcal{A}$ adaptively issues queries $q_1, ..., q_\lambda$, where query q_k is a trapdoor query (i, w). $\mathcal{C}$ responds by running algorithm Trap(sk_i, w) to derive $t_{i,w}$, and sends it to $\mathcal{A}$.*
- ***Guess:*** *$\mathcal{C}$ runs Trapdoor algorithm on input (i^*, w_b^*) for a random bit b to obtain t_{i^*,w_b^*}, where $w_b^* = w^*$ if $b = 0$, otherwise, it is a random keyword, and sends it to $\mathcal{A}$. $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$ for b and wins the game if $b = b'$.*

A SEMEKS scheme is trapdoor private if, for all PPT adversaries $\mathcal{A}$, for all sufficiently large λ, $\Pr[\text{win}_{\mathcal{A}}(\lambda, n)] < 1/2 + \text{negl}(\lambda)$.

4 Constructions

In this section, we give two constructions. For simplicity, we present the constructions assuming there is a single data owner; the scenario where there are multiple data owner can be solved similarly. For ease of exposition, we start with assuming that a centralized trusted third party initializes the system public keys; in Section 5.2, we describe how to eliminate the need of this trusted party. The data owner generates the secret key for each user, and then distributes the secret key to the user.

We adapt the coupling primitive in [2,3] carefully to the setting of multi-user keyword search where the data owner encrypts each file with a different key; different users are allowed to search different subset of (encrypted) files. Specifically, we couple the broadcast dimension (user index; each user is assigned an index) and the keyword dimension (keyword searchability). When user i wants to retrieve the documents that contain keyword w, user i generates trapdoor $t_{i,w}$ that has two dimensions: index i and keyword w. It binds both dimensions (index, keyword) to let the server search w in all the files that user i is allowed to access, and then retrieve the corresponding files. Both constructions use two useful cryptographic primitives: broadcast encryption from [9] and anonymous identity-based encryption from [11]. The broadcast encryption primitive provides constant size keyword ciphertexts and user secret keys. Specifically, it uses an aggregation method for ciphertexts that results in constant size ciphertexts (i.e., ciphertext size does not depend on the number of users). The anonymous encryption primitive provides anonymity of the keyword that is being encrypted. It uses anonymous encryption that does not reveal keyword from the ciphertext. Specifically, it uses linear splitting method on the random exponent values, which does not allow the adversary to do guessing attack (cross pairing attack).

The first construction uses a single server. We show that it satisfies data privacy but does not satisfy trapdoor privacy. The second construction (i.e., the main construction) is developed to address the problem. It uses two servers that do not collude. We show that it satisfies both data privacy and trapdoor privacy. At the end, we describe how to eliminate the need of having a single trusted party in our constructions.

4.1 First Construction

Setup(n, λ): Let $\mathbf{G}$ be a bilinear group of prime order p. The algorithm first picks a random generator $g \in \mathbf{G}$ and a random value $\alpha \in Z_p$. It computes $g_i = g^{\alpha^i} \in \mathbf{G}$ for $i = 1, 2, ..., n, n+2, ..., 2n$ (these values are generated by the centralized trusted third party). Next, it picks at random $\gamma, \beta \in Z_p$ and sets $v = g^{\gamma}, v' = g^{\beta} \in \mathbf{G}$. It then picks random elements $h_{0,1}, h_{0,2}, h_{1,1}, h_{1,2} \in \mathbf{G}$ and $a_1, b_1, a_2, b_2 \in Z_p$ (these values are picked by the data owner (DO)). The public

key is: $pk = (g, g_1, ..., g_n, g_{n+2}, ..., g_{2n}, v, v', h_{\ell,1}^{a_1}, h_{\ell,1}^{b_1}, h_{\ell,2}^{a_2}, h_{\ell,2}^{b_2})$, where $\ell = \{0, 1\}$. The secret key of the DO is $sk_{DO} = (\gamma, \beta, a_1, b_1, a_2, b_2)$. The algorithm outputs pk and sk_{DO}. For users' secret keys, the DO chooses random $\rho_{i1}, \rho_{i2}, \rho'_{i1}, \rho'_{i2}$ for user i and computes secret key for user i as follows:

$d_{i,1} = g_i^{\gamma}$, $d_{i,2} = g^{a_1\rho_{i1}}$, $d_{i,3} = g^{a_1\rho'_{i1}}$, $d_{i,4} = g^{a_2\rho_{i2}}$,
$d_{i,5} = g^{a_2\rho'_{i2}}$, $d_{i,6} = g^{b_1\rho_{i1}}$, $d_{i,7} = g^{b_1\rho'_{i1}}$, $d_{i,8} = g^{b_2\rho_{i2}}$,
$d_{i,9} = g^{b_2\rho'_{i2}}$, $d_{i,10} = h_{0,1}^{a_1b_1\rho_{i1}} h_{0,2}^{a_2b_2\rho_{i2}}$,
$d_{i,11} = h_{0,1}^{a_1b_1\rho'_{i1}} h_{0,2}^{a_2b_2\rho'_{i2}}$, $d_{i,12} = h_{1,1}^{a_1b_1\rho_{i1}} h_{1,2}^{a_2b_2\rho_{i2}}$,
$d_{i,13} = h_{1,1}^{a_1b_1\rho'_{i1}} h_{1,2}^{a_2b_2\rho'_{i2}}$, $d_{i,14} = g_i^{\beta}$.

These values are given to user i via a secure channel. Specifically, $sk_i = (d_{i,1}, \ldots, d_{i,14})$.

Enc(pk, w, S_k, M_k): The DO picks random values $t, t', t_1, t_2 \in Z_p$ for keyword w from F_k, and computes the followings:

$K = e(g_{n+1}, g)^t$, $K' = e(g_{n+1}, g)^{t'}$, $hdr_{k,1} = (h_{0,1}^{a_1}(h_{1,1}^{a_1})^w)^{t_1}$,
$hdr_{k,2} = (h_{0,1}^{b_1}(h_{1,1}^{b_1})^w)^{t-t_1}$, $hdr_{k,3} = (h_{0,2}^{a_2}(h_{1,2}^{a_2})^w)^{t_2}$,
$hdr_{k,4} = (h_{0,2}^{b_2}(h_{1,2}^{b_2})^w)^{t-t_2}$, $hdr_{k,5} = g^t$, $hdr_{k,6} = (v\prod_{j\in S_k} g_{n+1-j})^t$,
$hdr_{k,7} = K$, $hdr_{k,8} = g^{t'}$, $hdr_{k,9} = (v'\prod_{j\in S_k} g_{n+1-j})^{t'}$, $hdr_{k,10} = K'M_k$.

Let the first part of the ciphertext $C_k = (hdr_{k,1}, \ldots, hdr_{k,7})$. Let the second part of the ciphertext $C''_k = (hdr_{k,8}, hdr_{k,9}, hdr_{k,10})$

Trap$(d_{i,1}||....||d_{i,13}, w)$: User i picks $r, r' \in Z_p$ and computes $t_{i,w}$ as

$$
\begin{aligned}
tr_1 &= d_{i,1} d_{i,10}^{r} d_{i,11}^{r'} \left(d_{i,12}^{r} d_{i,13}^{r'}\right)^{w}, \\
tr_2 &= d_{i,2}^{r} d_{i,3}^{r'}, \\
tr_3 &= d_{i,4}^{r} d_{i,5}^{r'}, \\
tr_4 &= d_{i,6}^{r} d_{i,7}^{r'}, \\
tr_5 &= d_{i,8}^{r} d_{i,9}^{r'}.
\end{aligned}
$$

Test$(pk, S_k, t_{i,w}, C_k)$: The server checks if

$$hdr_{k,7} \stackrel{?}{=} \frac{e(g_i, hdr_{k,6})e(hdr_{k,1}, tr_4)T}{e(tr_1 \prod_{j\in S_k, i\neq j} g_{n+1-j+i}, hdr_{k,5})},$$

where $T = e(hdr_{k,3}, tr_5)e(hdr_{k,2}, tr_2)e(hdr_{k,4}, tr_3)$. If the equality holds, then the test result $b = 1$. Otherwise, $b = 0$.

Dec$(pk, d_{i,14}, C''_k, S_k)$: Once the server outputs $b = 1$ from the **Test** algorithm, he sends S_k, C''_k to user i (we call this process as *download*). Then, user i does the decryption in the same way as that in [9] to recover first K' then extracts M_k by computing $M_k = \frac{K'M_k}{K'}$.

It is easy to see that the keyword ciphertext does not reveal w by using cross pairing. The data privacy is achieved by using linear splitting method that is introduced in [11] for ciphertexts. The idea in [11] is to use different random blind values in ciphertexts. The trapdoor privacy is achieved by blinding the secret key of a user. This scheme is correct as follows:

$(pk, sk_{DO}, sk_i = (d_{i,1}||\ldots||d_{i,14})_{i=1,..n})) \leftarrow \mathbf{Setup}(n, \lambda)$,
$(C_k, C_k'') = (hdr_{k,j})_{j=[1,10]} \leftarrow \mathbf{Enc}(pk, w, S_k, M_k)$,
$t_{i,w} = (tr_l)_{l=[1,5]} \leftarrow \mathbf{Trap}(sk_i = (d_{i,1}||\ldots||d_{i,13}), w)$,
If $i \in S_k$, $b \leftarrow \mathbf{Test}(pk, t_{i,w}, C_k, S_k)$,
Otherwise, $\perp \leftarrow \mathbf{Test}(pk, t_{i,w}, C_k, S_k)$,
If $b = 1$, $M_k \leftarrow \mathbf{Dec}(pk, d_{i,14}, S_k, C_k'')$.

The first construction does not have trapdoor privacy. The first reason is that the adversary (i.e., the server) can extract the keyword from the generated trapdoor. The problem happens because user i does not blind his first part of the secret key $d_{i,1}$ when he generates the first part of the trapdoor tr_1. The attack basically occurs when given a trapdoor for a keyword w from user i, $t_{i,w} = (tr_1, tr_2, tr_3, tr_4, tr_5)$, the server picks a keyword w^* and checks if

$$\frac{e(tr_1, g)}{e(h_{0,1}^{b_1}, tr_2)e(h_{0,2}^{b_2}, tr_3)e(g_i, g^{\gamma})} = e(h_{1,1}^{a_1}, tr_4)^{w^*} e(h_{1,2}^{a_2}, tr_5)^{w^*}.$$

If the above equality holds, the server concludes that $w = w^*$. The second reason is going to be explained in Section 5.1.

To counter the above attack, we need to blind the first part of the secret key of a user, which however breaks the BGW decryption process since it needs the first part of the secret key not to be blinded. We solve the above issue in the following construction by using two servers $\mathcal{S}_{main}$ and $\mathcal{S}_{aid}$. In our new model, $\mathcal{S}_{aid}$ is trusted while $\mathcal{S}_{main}$ is the semi-honest adversary. In trapdoor phase, a user chooses three random values r, r', r'' and generates the trapdoor using r, r', r''. The user uses r'' to blind the first part of secret key d_1 and uses r, r' to blinds other parts of the secret keys (d_j, where $j = 2, .., 13$). Then, the user sends a random value r_2 with trapdoor to $\mathcal{S}_{main}$ and another random value r_1 to $\mathcal{S}_{aid}$. Here, r'' can be thought as a function of r_1, r_2: $f(r_1, r_2) = r''$. In our construction, the function f simply takes r_1, r_2 and outputs $r'' = r_1 + r_2$. $\mathcal{S}_{aid}$ stores the values $C_k' = hdr_{k,5}, hdr_{k,6}, hdr_{k,7}, S_k$ while $\mathcal{S}_{main}$ stores $(C_k, C_k'') = (hdr_{k,1}, hdr_{k,2}, hdr_{k,3}, hdr_{k,4}, hdr_{k,5}, hdr_{k,6}, hdr_{k,7}, hdr_{k,8}, hdr_{k,9}, hdr_{k,10}, S_k)$ for file F_k. Once the user sends trapdoor and random values to the servers, $\mathcal{S}_{aid}$ first computes $f_1(r_1, C_k')$ and sends it to $\mathcal{S}_{main}$. Then $\mathcal{S}_{main}$ internally checks if the keyword appears in the ciphertext by using $t_{i,w}, f_2(r_2, C_k), f_1(r_1, C_k'), C_k, S_k$. If so, $\mathcal{S}_{main}$ sends C_k'' and S_k to the user. The user decrypts it and recovers the plaintext file k where the keyword appears. We illustrate the query process in Fig. 1.

Remark 1. The first construction can be improved in the size of system public key, the size of user secret key, the size of trapdoor and the size of keyword ciphertext but the improved version of the first construction does not allow the adversary to make trapdoor queries. In the interest of space, we do not give further information about the improved version.

4.2 Main Construction

This construction has two servers. It is as follows (illustrated in Fig. 1).

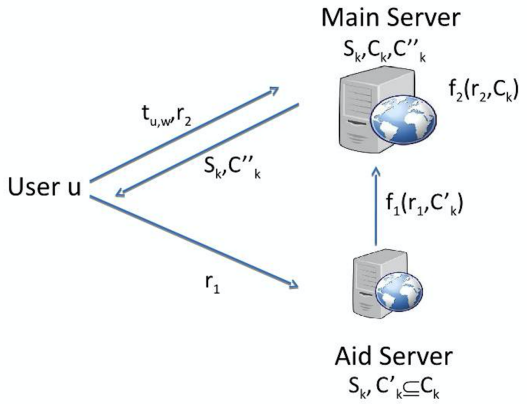

Fig. 1. Illustration of the secure and efficient construction.

Setup(n, λ): This algorithm is the same as that in the first construction in Section 4.1. It outputs system public key pk and user secret key sk_i for user $i = 1, \ldots, n$.

Enc(pk, w, S_k, M_k): This algorithm is the same as that in the first construction in Section 4.1 and outputs $hdr_{k,1} \ldots hdr_{k,10}$. Let the first part of the ciphertext $C_k = (hdr_{k,1}, \ldots, hdr_{k,7})$. Let the second part of the ciphertext $C''_k = (hdr_{k,8}, hdr_{k,9}, hdr_{k,10})$. In addition, let $C'_k = (hdr_{k,5}, hdr_{k,6}, hdr_{k,7})$. The main server $\mathcal{S}_{main}$ stores C_k, C''_k and S_k. The aid server $\mathcal{S}_{aid}$ stores C'_k and S_k.

Trap$(d_{i,1}||....||d_{i,13}, w)$: User i picks $r, r', r'' \in Z_p$ and computes $t_{i,w}$ as

$$
\begin{aligned}
tr_1 &= d_{i,1}^{r''} d_{i,10}^{r} d_{i,11}^{r'} \left(d_{i,12}^{r} d_{i,13}^{r'} \right)^{w}, \\
tr_2 &= d_{i,2}^{r} d_{i,3}^{r'}, \\
tr_3 &= d_{i,4}^{r} d_{i,5}^{r'}, \\
tr_4 &= d_{i,6}^{r} d_{i,7}^{r'}, \\
tr_5 &= d_{i,8}^{r} d_{i,9}^{r'}, \\
tr_6 &= g_{n+1+i}^{r''}.
\end{aligned}
$$

He sends $t_{i,w} = (tr_1, \ldots, tr_6)$ and r_2 to $\mathcal{S}_{main}$, and sends r_1 to $\mathcal{S}_{aid}$.

Test$(pk, S_k, f_2(r_2, C_k), f_1(r_1, C'_k), t_{i,w}, C_k)$: Both servers compute $A_k = \frac{e(g_i, hdr_{k,6})}{hdr_{k,7} e\left(\prod_{j \in S_k, i \neq j} g_{n+1-j+i}, hdr_{k,5}\right)}$ for file k. The aid server sends $f_1(r_1, C'_k) = A_k^{r_1}$ to $\mathcal{S}_{main}$. $\mathcal{S}_{main}$ then computes $f_2(r_2, C_k) f_1(r_1, C'_k) = A_k^{r_2} A_k^{r_1} = A_k^{r}$ and checks if $\frac{e(tr_1, hdr_{k,5})}{e(hdr_{k,1}, tr_4) T} = A_k^{r}$, where $T = e(hdr_{k,3}, tr_5) e(hdr_{k,2}, tr_2) e(hdr_{k,4}, tr_3)$. If the equality holds, then the test result $b = 1$. Otherwise, $b = 0$.

Dec$(pk, d_{i,14}, C''_k, S_k)$: Once the server outputs $b = 1$ from the **Test** algorithm, he sends S_k, C''_k to user i (we call this process as *download*). Then, user i does the decryption in the same way as that in [9] to recover first K' then extracts M_k by computing $M_k = \frac{K' M_k}{K'}$.

Remark 2. The encryption algorithm takes fresh random values in Z_p to encrypt each keyword and message file.

5 Security

Proving Security: We prove security using a hybrid experiment as that in [11]. Let $[hdr_{0,1}, hdr_{0,2}, hdr_{0,3}, hdr_{0,4}, hdr_{0,5}, hdr_{0,6}, hdr_{0,7}]$ denote the challenge ciphertext for keyword w^* and user set S_0 that are given to the adversary during a real attack. Additionally, let R, R' be two random elements of $\mathbf{G}$. We define the following hybrid games which differ on what challenge ciphertext is given by a simulator $\mathcal{SIM}$ to the adversary:
Γ_0: The challenge ciphertext is $C_0 = [hdr_{0,1}, hdr_{0,2}, hdr_{0,3}, hdr_{0,4}, hdr_{0,5}, hdr_{0,6}, hdr_{0,7}]$.
Γ_1: The challenge ciphertext is $C_1 = [hdr_{0,1}, R, hdr_{0,3}, hdr_{0,4}, hdr_{0,5}, hdr_{0,6}, hdr_{0,7}]$.
Γ_2: The challenge ciphertext is $C_2 = [hdr_{0,1}, R, hdr_{0,3}, R', hdr_{0,5}, hdr_{0,6}, hdr_{0,7}]$.

We remark that the challenge ciphertext in Γ_2 leaks no information about the keyword since it is composed of seven random group elements, whereas in Γ_0 the challenge is well formed. We show that the transitions from Γ_0 to Γ_1 and Γ_1 to Γ_2 are all computationally indistinguishable.

Since we use two servers, we slightly change the definition of keyword privacy game in Section 3. During the game, the adversary makes encryption queries since $\mathcal{S}_{aid}$ is trusted and controlled by the simulator. The adversary is not able to upload any ciphertext that he wants. We restrict the adversary.

Theorem 2 (Keyword Privacy). *The main construction of the SEMEKS scheme has keyword privacy under Decision Linear Assumption.*

Proof. Suppose the existence of an adversary $\mathcal{A}$ that distinguishes between the two games (Γ_0 and Γ_1) with a non-negligible advantage ϵ. Then we construct $\mathcal{SIM}$ that wins the Decision Linear game as follows. $\mathcal{SIM}$ takes in a decision linear instance $g, g^{z_1}, g^{z_2}, g^{z_1 z_3}, g^{z_2 z_4}, Z$, where Z is either $g^{z_3+z_4}$ or random in $\mathbf{G}$ with equal probability. For convenience, we rewrite this as $g, g^{z_1}, g^{z_2}, g^{z_1 z_3}, Y, g^t$ for t such that $g^t = Z$, and consider the task of deciding whether $Y = g^{z_2(t-z_3)}$. $\mathcal{SIM}$ plays the game in the following stages.

- **Init**: $\mathcal{A}$ gives $\mathcal{SIM}$ the challenge keyword w^* and the challenge set S_0.
- **Setup**: The simulator first chooses random exponents $\alpha, \gamma, \zeta, a_2, b_2$. It uses the same g as in the decision linear instance, and sets $g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}, v$, where $g_i = g^{\alpha^i}$, $v = g^\gamma$, $h_{0,1} = g^{-w^*\zeta+\zeta}$, $h_{1,1} = g^\zeta$,$h_{0,2} = g^{z_2(-w^*\zeta)} g^\zeta$, $h_{1,2} = g^{z_2\zeta}$. Next the simulator sets $h_{0,1}^{a_1} \leftarrow (g^{z_1})^{-w^*\zeta+\zeta}$, $h_{1,1}^{a_1} \leftarrow (g^{z_1})^\zeta$, $h_{0,1}^{b_1} \leftarrow (g^{z_2})^{-w^*\zeta+\zeta}$, $h_{1,1}^{b_1} \leftarrow (g^{z_2})^\zeta$,$h_{0,2}^{a_2} \leftarrow g^{a_2 z_2(-w^*\zeta)} g^{a_2\zeta}$, $h_{1,2}^{a_2} \leftarrow g^{a_2 z_2 \zeta}$, $h_{0,2}^{b_2} \leftarrow g^{b_2 z_2(-w^*\zeta)} g^{b_2\zeta}$, $h_{1,2}^{b_2} \leftarrow g^{b_2 z_2 \zeta}$.

The system public key is $pk = g, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}, v, h_{0,1}^{a_1}, h_{1,1}^{a_1}, h_{0,1}^{b_1}, h_{1,1}^{b_1}$, $h_{0,2}^{a_2}$, $h_{1,2}^{a_2}$, $h_{0,2}^{b_2}$,$h_{1,2}^{b_2}$ and $\mathcal{SIM}$ gives the public key to $\mathcal{A}$ and $\mathcal{S}_{aid}$.

- **Query**(S_ℓ, w_ℓ): To answer encryption query for (S_ℓ, w_ℓ) where $w_\ell \neq w^*$, the simulator picks ℓ, ℓ_1, ℓ_2, and computes $hdr_{k,1}, \ldots, hdr_{k,7}$ as
  $K = e(g_{n+1}, g)^\ell$, $\quad hdr_{k,1} = (h_{0,1}^{a_1}(h_{1,1}^{a_1})^{w_\ell})^{\ell_1}$,
  $hdr_{k,2} = (h_{0,1}^{b_1}(h_{1,1}^{b_1})^{w_\ell})^{\ell-\ell_1}$, $\quad hdr_{k,3} = (h_{0,2}^{a_2}(h_{1,2}^{a_2})^{w_\ell})^{\ell_2}$,
  $hdr_{k,4} = (h_{0,2}^{b_2}(h_{1,2}^{b_2})^{w_\ell})^{\ell-\ell_2}$, $\quad hdr_{k,5} = g^\ell$, $\quad hdr_{k,6} = (v\prod_{j\in S_\ell} g_{n+1-j})^\ell$,
  $hdr_{k,7} = K$ and gives them to $\mathcal{A}$ ($\mathcal{S}_{main}$) and gives $hdr_{k,5}, hdr_{k,6}, hdr_{k,7}, S_\ell$ to $\mathcal{S}_{aid}$.
- **Query**(i, w): To answer trapdoor queries for (i, w) where $i \notin S_0$ and $w \neq w^*$, the simulator picks $\rho_{i1}, \rho'_{i1}, \rho_{i2}, \rho'_{i2}, r, r', r''$, and computes $t_{i,w} = (tr_1, tr_2, tr_3, tr_4, tr_5, tr_6)$ as
  $tr_1 = g_i^{r''\gamma}(h_{0,2}h_{1,2}^w)^{a_2b_2Y}(g^{\frac{-z_1Xz}{w-w^*}})(g^{-z_1Xz})$, $tr_2 = g^{z_1X}, tr_3 = g^{z_2X}, tr_4 = g^{a_2Y}g^{\frac{-z_1X}{b_2}}g^{\frac{-z_1X}{b_2(w-w^*)}}$, $tr_5 = g^{b_2Y}g^{\frac{-z_1X}{a_2}}g^{\frac{-z_1X}{a_2(w-w^*)}}, tr_6 = g_{n+1+i}^{r''}$, where $X = \rho_{i1}r + \rho'_{i1}r'$, $Y = \rho_{i2}r + \rho'_{i2}r'$. Then, he gives $tr_1, tr_2, tr_3, tr_4, tr_5, tr_6, r_2$ to $\mathcal{A}$ and r_1 to $\mathcal{S}_{aid}$.
- **Guess**: The simulator responds with a challenge ciphertext for the keyword w^* and the set S_0. Assume $t_1 = z_3$. The simulator picks random $t_2 \in Z_p$. To proceed, the simulator outputs the ciphertext as $hdr_{0,1} = (g^\zeta)^{(t_1z_1)}, hdr_{0,2} = Y^\zeta, hdr_{0,3} = (g^\zeta)^{(t_2a_2)}, hdr_{0,4} = Z^{\zeta b_2}g^{-\zeta b_2t_2}, hdr_{0,5} = Z = g^t, hdr_{0,6} = \left(v\prod_{j\in S_0} g_{n+1-j}\right)^t, hdr_{0,7} = K = e(g_{n+1}, g)^t$.
  If $Y = g^{z_2(t-z_3)}$ so $Z = g^t$, then all parts of the challenge are well formed and the simulator simulates game Γ_0. If instead Y is independent of z_1, z_2, t, t_1, t_2, which happens when Z is random, then the simulator emulates the game Γ_1.
- **Output**: $\mathcal{A}$ outputs a bit b to guess which hybrid game the simulator has been playing. To conclude, the simulator forwards b as its own answer in the Decision Linear game.

Restriction: $\mathcal{S}_{aid}$ does not use the public key g_i that is asked in trapdoor query to compute function $f(r_1, C')$. The function should be independent of g_i since the index i does not appear in the set S_0.

Analysis: Since the challenge ciphertext is independent of w^*, the adversary's best success probability is $1/2$ when the adversary $\mathcal{A}$ gets Γ_1 as challenge ciphertext. The success probability is $\Pr[\text{win}_{\mathcal{A}}(\lambda, n)] < 1/2 + \epsilon$ when $\mathcal{A}$ gets Γ_0 as challenge ciphertext. So, $\mathcal{SIM}$ breaks the Decision Linear assumption with probability $|\Pr(\mathcal{A}(\Gamma_0) = 1) - \Pr(\mathcal{A}(\Gamma_1) = 1| = 1/2 + \epsilon - 1/2 = \epsilon$, which is non-negligible.

□

Remark 3. The indistinguishability of the hybrid games Γ_1 and Γ_2 can be shown similarly by adjusting the parameters as $a_2 = z_1$, $b_2 = z_2$.

Theorem 3 (File Privacy). *The first construction of SEMEKS has file privacy under n-DBDHE assumption.*

Proof. The other parts of the (file) ciphertexts, $(g^{t'}, (v'(\prod_{j \in S_0} g_{n+1-j}))^{t'}, K'M_0)$ are just for downloading process (independent of the first part of the keyword ciphertext and other public key values that are formed by $a_1, a_2, b_1, b_2, \gamma$ and $h_{0,1}, h_{1,1}, h_{0,2}, h_{1,2}$) when the searched keyword matches the first part of the keyword ciphertexts. These parts, C_0'', can be simulated in the same way as in [9]. In the interests of space, we do not give concrete proof.

As a proof sketch, once $\mathcal{SIM}$ gets n-DBDHE parameters, he will set pk, $hdr_{0,8}$, $hdr_{0,9}$, v' and user secret keys $d_{j,14}$, where $j \notin S_0$, in the same way as in [9]. Then, he follows the same game steps (Challenge and Guess) as those in [9]. □

5.1 Trapdoor Privacy

Formalizing a security model for trapdoor privacy is challenging since an adversary can use the public resource and/or provided secret information (trapdoor) to verify if the given trapdoor is generated under a specific keyword. This is a serious problem if a scheme is built in a public key setting. It is because the adversary can encrypt any keyword, then he can verify if the encrypted keyword and the given trapdoor are generated under the same keyword. The solution for this is to restrict the adversary to avoid generating any ciphertext for a set of users to verify if the given trapdoor and the generated ciphertext match under the same keyword.

Our Solution: Since we use a two-server solution for trapdoor privacy, we change the definition of trapdoor privacy in Section 3. In the new model, we are going to have some query restrictions.

Restriction: For a challenge trapdoor (i^*, w^*) in the trapdoor privacy game, the restriction on $\mathcal{A}$ for encryption query $\text{Enc}(pk, w, S_j)$: if $i^* \in S_j$ or $w = w^*$, the simulator outputs $\perp$. It means that the adversary is not able to ask the challenger an encryption query where the challenge user index is in the given set S_j or the challange keyword. One can think that the adversary can encrypt whatever he wants to encrypt since he has the system public key pk. However, in our model $\mathcal{S}_{aid}$ is trusted and is controlled by the simulator. So, the adversary is not able to upload any keyword ciphertext to $\mathcal{S}_{aid}$. But he can encrypt keywords on his will offline. It means that he can encrypt keywords and see the ciphertext but he is not able to use them in test algorithm since he needs $\mathcal{S}_{aid}$ to send partial trapdoor information. Another restriction is that $\mathcal{S}_{aid}$ does not use the public key of the challenge user (g_i) to compute function $f(r_1, C')$. The function should be independent of g_i.

Theorem 4 (Trapdoor Privacy). *The construction of SEMEKS is trapdoor private under n-DDHI assumption.*

In the interest of space, we leave the proof to Appendix A.3.

5.2 Eliminating Single Trusted Party

The two constructions above use a single trusted party to generate the public key pk. The single trusted party can be a manager or a DO. In either case, a DO

generates secret keys for the users that are eligible to do keyword search on the DO's files and distributes the secret keys to the users. A user's secret key is hence known by the DO, which is not desirable since a user might want her secret key to be only known by herself. Furthermore, when there are multiple DOs and a user wants to search multiple DOs' data files, each DO needs to generate and send a secret key to the user, and the user needs to generate a trapdoor that is linear in the number of DOs. This results in a scalability problem. In addition, if there are multiple DOs, each setting her own public key which is of size $O(n)$, the system total public key size is going to be $O(n^2)$ for n DOs. This results in another scalability problem.

To address the above issues, we can use the recently proposed *distributed parameter generation protocol* [22]. In [22], n parties can jointly generate system public key and user secret keys that eliminates the need of having a single trusted authority. These n parties can be all DOs that share their data files with each other (group data sharing), or a single DO that share his files with $n-1$ users. Since most parts of the system public key are in the from of *n-BDHE* parameters, they can be generated straightforwardly from [22]. The remaining part of the values $h_{0,1}^{a_1}, h_{1,1}^{a_1}, h_{0,1}^{b_1}, h_{1,1}^{b_1}, h_{0,2}^{a_2}, h_{1,2}^{a_2}, h_{0,2}^{b_2}, h_{1,2}^{b_2}$ can be distributively generated by applying $\mathcal{DKG}$ and $\mathcal{REC}$ a couple of times. The secret keys of the users can be distributively generated by applying $\mathcal{DKG}, \mathcal{REC}$ and $\mathcal{RECSQ}$ sub-protocols. Since these are straightforward, we do not provide the constructions explicitly.

6 Conclusion

We have proposed a secure and efficient encrypted keyword search (SEMEKS) scheme for multi-user data sharing. In the scheme, an authorized user uses a single trapdoor to search all files that he is authorized to search. In addition, for keyword ciphertexts of a file, the network bandwidth usage and the storage required at the server does not depend on the number of authorized users that can search that file. We have also performed rigorous security analysis to show that SEMEKS has trapdoor privacy and data privacy.

As future work, we will investigate how to use a single server to achieve secure and efficient keyword search for data sharing. In addition, we want to support boolean and not conjunctive keyword queries and update users set efficiently (adding or removing users from pre-defined authorized set). It is also interesting to evaluate the overall performance of our scheme such as communication and computation complexities related to trapdoor generation, keyword search, encryption and decryption.

References

1. M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *CRYPTO*, 2005.

2. N. Attrapadung. *Unified Frameworks for Practical Broadcast Encryption and Public Key Encryption with High Functionalities*. PhD thesis, University of Tokyo, 2007.
3. N. Attrapadung, J. Furukawa, and H. Imai. Forward-secure and searchable broadcast encryption with short ciphertexts and private keys. In *ASIACRYPT*, 2006.
4. F. Bao, R. H. Deng, X. Ding, and Y. Yang. Private query on encrypted data in multi-user settings. In *ISPEC*, 2008.
5. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, 2007.
6. D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, 2005.
7. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO*, 2004.
8. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *EUROCRYPT*, 2004.
9. D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.
10. D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, 2007.
11. X. Boyen and B. Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, 2006.
12. D. Cash, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, 2013.
13. Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, 2005.
14. C.-K. Chu, S. S. M. Chow, W.-G. Tzeng, J. Zhou, and R. H. Deng. Key-aggregate cryptosystem for scalable data sharing in cloud storage. *IEEE Transanctions on Parallel and Distributed Systems*, 2014.
15. B. Cui, Z. Liu, and L. Wang. Key-aggregate searchable encryption (KASE) for group data sharing via cloud storage. *IEEE Transactions on Computers,*, 2015.
16. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *CCS*, 2006.
17. C. Dong, G. Russello, and N. Dulay. Shared and searchable encrypted data for untrusted servers. In *Proceeedings of the 22Nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, 2008.
18. S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner. Rich queries on encrypted data: Beyond exact matches. In *ESORICS*, 2015.
19. E.-J. Goh. Secure indexes. Cryptology eprint archive, report 2003/216, 2003.
20. S. Jarecki, C. S. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *CCS*, 2013.
21. S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS*, 2012.
22. A. Kiayias, O. Oksuz, and Q. Tang. Distributed parameter generation for bilinear Diffie-Hellman exponentiation and applications. In *ISC*, 2015.
23. K. Liang and W. Susilo. Searchable attribute-based mechanism with efficient data sharing for secure cloud storage. *IEEE Transactions on Information Forensics and Security*, 2015.
24. Z. Liu, J. Li, X. Chen, J. Yang, and C. Jia. TMDS: Thin-model data sharing scheme supporting keyword search in cloud storage. In *ACISP*, 2014.
25. R. A. Popa and N. Zeldovich. *Multi Key Searchable Encryption*. https://people.csail.mit.edu/nickolai/papers/popa-multikey-eprint.pdf, 2013.

26. V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.
27. D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, 2000.
28. Q. Tang. Nothing is for free: Security in searching shared and encrypted data. *Transaction on Information Forensics and Security*, 2014.
29. C. Van Rompay, R. Molva, and M. Onen. Multi-user searchable encryption in the cloud. In *ISC*, 2015.
30. Q. Zheng, S. Xu, and G. Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *INFOCOM*, 2014.

A Appendix

A.1 Vulnerability of the Scheme in [15]

Cui et al. [15] proposed a scheme that provides an aggregation method on files for a user. Basically, each user's secret key is mapped to an aggregated number of files. With this aggregation function, in their scheme, a file size does not depend on the number users. In addition, in their scheme, keyword ciphertext size is constant. We argue that their scheme is vulnerable to cross pairing attacks. Specifically, (1) the adversary (server) is able to extract keywords from the ciphertext, and (2) the adversary captures the secret key of any user so the adversary is able to build trapdoors to search any keyword as an authorized user. The first attack (cross-pairing or dictionary or guessing) is to the ciphertexts as follows. Since each file j is encrypted with a single encryption key t_j, the ciphertexts of the keywords are the form of, $C_{w_1} = \frac{e(H(w_1),g)^{t_j}}{e(g_n,g_1)^{t_j}}$, $C_{w_2} = \frac{e(H(w_2),g)^{t_j}}{e(g_n,g_1)^{t_j}}$. Given $C_{w_1}, C_{w_2}, C_1 = g^{t_j}$, $C_2 = (vg_j)^{t_j}$, the server (adversary) picks two keywords w_1^*, w_2^* and checks if $\frac{e(H(w_1^*),g)^{t_j}}{C_{w_1}} = e(g_n, g_1)^{t_j} = \frac{e(H(w_1^*),g)^{t_j}}{C_{w_2}}$. If so, the server concludes that $w_1^* = w_1, w_2^* = w_2$. Moreover, the adversary recovers $e(g_n, g_1)^{t_j}$ for file j.

Another weakness of the scheme in [15] happens in producing a trapdoor. Basically, the DO computes a secret key for a user by computing $k_{agg} = \prod_{j \in S} g_{n+1-j}^{\gamma}$, where $\gamma \in Z_p$ is the secret key of the DO, and any subset $S \subseteq \{1, .., n\}$ which contains the indices of documents that the user is authorized to search. Once the user gets the secret key, he makes a query for keyword w_1 by computing $Tr_1 = k_{agg} H(w_1)$. The user sends (Tr_1, S) to the could server. If the same user makes another query for keyword w_2, he sends $(Tr_2 = k_{agg} H(w_2), S)$ to the server. The problem is that the server can choose two keywords w^*, w^{**} and computes $H(w^*), H(w^{**})$. Then, the server checks if $Tr_1 H(w^*) = Tr_2 H(w^{**})$ or $Tr_1 H(w^{**}) = Tr_2 H(w^*)$. If the first equality holds, the server concludes $H(w_1) = H(w^{**})$ and $H(w_2) = H(w^*)$ while the second equality holds, the server concludes $H(w_1) = H(w^*)$ and $H(w_2) = H(w^{**})$. Either case, the server obtains secret key of the user k_{agg}. This is the very crucial information because the server makes query as if an authorized user.

A.2 Generic Security of n-DDHI Assumption

In the generic group model, elements of $\mathbf{G}$ and $\mathbf{G}'$ appear to be encoded as unique random strings, so that no property other than equality can be directly tested by the adversary $\mathcal{A}$. Three oracles are assumed to perform operations between group elements, such as computing the group action in each of the two groups $\mathbf{G}, \mathbf{G}'$ and the bilinear pairing $e : \mathbf{G} \times \mathbf{G} \to \mathbf{G}'$. The opaque encoding of the elements of $\mathbf{G}$ is modeled as injective functions ξ, ξ' that are chosen at random.

Let $\xi : Z_p \to \Xi$, where $\Xi \subset \{0,1\}^*$, which maps all $a \in Z_p$ to the string representation $\xi(g^a)$ of $g^a \in \mathbf{G}$. We similarly define $\xi' : Z_p \to \Xi'$ for $\mathbf{G}'$. The attacker $\mathcal{A}$ communicates with the oracles using the ξ-representations of the group elements only.

Let $\alpha, \gamma, z \in Z_p^*$, $T_0 \leftarrow g^{\frac{z\gamma}{\alpha^{n+1}}}$, where $h = g^z$, $T_1 \leftarrow g^r$, and $d \leftarrow \{0,1\}$. We show that no generic algorithm $\mathcal{A}$ that is given the encodings of $g, h = g^z, g_1, .., g_n, g_{n+2}, .., g_{2n}, g'$, where $g_i = g^{\alpha^i}$, $g' = g^\gamma$ and makes up to q oracle queries can guess the value of d with probability great than $\frac{1}{2} + O(\frac{q^2 n^3}{p})$.

Proof of Theorem 1. Consider an algorithm $\mathcal{B}$ that plays the following game with $\mathcal{A}$. $\mathcal{B}$ maintains two lists of pairs, $L_1 = \{(F_{1,i}, \xi_{1,i}) : i = 0, ..., \tau_1 - 1\}, L_T = \{(F_{T,i}, \xi'_{T,i}) : i = 0, ..., \tau_2 - 1\}$, under the invariant that, at step τ in the game, $\tau_1 + \tau_T = \tau + 2n + 2$. Here, the $F_{*,*} \in Z_p[A, \Gamma, Z, T_0, T_1]$ are polynomials in the indeterminates A, Γ, Z, T_0, T_1 with coefficients in Z_p. The $\xi'_{*,*} \in \{0,1\}^*$ are arbitrary distinct strings.

The lists are initialized at step $\tau = 0$ by initializing $\tau_1 \leftarrow 2n + 2, \tau_T \leftarrow 0$, and setting $F_{1,0} = 1$, $F_{1,1} = A$, $F_{1,2} = A^2$,..., $F_{1,n} = A^n$, $F_{1,(n+2)} = A^{n+2}$,..., $F_{1,2n} = A^{2n}$, $F_{1,2n+1} = \Gamma$, $F_{1,2n+2} = z$, $F_{1,2n+3} = T_0$, $F_{1,2n+4} = T_1$. The corresponding strings are set to arbitrary distinct strings in $\{0,1\}^*$ (Here, $F_{1,n+1}$ is skipped).

We may assume that $\mathcal{A}$ only makes oracle queries on strings previously obtained from $\mathcal{B}$, since $\mathcal{B}$ can make them arbitrarily hard to guess. We note that $\mathcal{B}$ can determine the index i of any given string $\xi_{1,i}$ in L_1 (resp. $\xi'_{T,i} \in L_T$), where ties between multiple matches are broken arbitrarily.

$\mathcal{B}$ starts the game by providing $\mathcal{A}$ with the encodings $\xi_{1,0}, \xi_{1,1}, \xi_{1,2}, \ldots, \xi_{1,n}$, $\xi_{1,(n+2)}, \ldots, \xi_{1,2n}, \xi_{1,(2n+1)}, \xi_{1,2n+2}, \xi_{1,2n+3}, \xi_{1,2n+4}$. The simulator $\mathcal{B}$ responds to algorithm $\mathcal{A}$'s queries as follows.

Group action. Given a multiply/divide selection bit and two operands $\xi_{1,i}$ and $\xi_{1,j}$ with $0 \le i, j < \tau_1$, compute $F_{1,\tau_1} \leftarrow F_{1,i} \mp F_{1,j}$ depending on whether a multiplication or a division is requested. If $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$, set $\xi_{1,\tau_1} \leftarrow \xi_{1,l}$; otherwise, set ξ_{1,τ_1} to a string in $\{0,1\}^*$ distinct from $\xi_{1,0}, ..., \xi_{1,\tau_1-1}$. Add $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to the list L_1 and give ξ_{1,τ_1} to $\mathcal{A}$, then increment τ_1 by one. Group action queries in $\mathbf{G}'$ are treated similarly.

Pairing. Given two operands $\xi_{1,i}$ and $\xi_{1,j}$ with $0 \le i, j < \tau_1$, compute the product $F_{T,\tau_T} \leftarrow F_{T,i} F_{T,j}$. If $F_{T,\tau_T} = F_{T,l}$ for some $l < \tau_T$, set $\xi'_{T,\tau_T} \leftarrow \xi'_{T,l}$; otherwise, set ξ'_{T,τ_T} to a string in $\{0,1\}^* \setminus \{\xi'_{T,0}, ..., \xi'_{T,\tau_T-1}\}$. Add $(F_{T,\tau_T}, \xi'_{T,\tau_T})$ to the list L_T , and give ξ'_{T,τ_T} to $\mathcal{A}$, then increment τ_T by one.

Observe that at any time in the game, the total degree of any polynomial in each of the two lists is bounded as follows: $deg(F_{1,i}) \leq 2n$, $deg(F_{T,i}) = 4n$. After at most q queries, $\mathcal{A}$ terminates and returns a guess $d' \in \{0,1\}$. At this point $\mathcal{B}$ chooses random $\alpha, \gamma, z \leftarrow Z_p$. Consider $t_d \leftarrow \frac{z\gamma}{\alpha^{n+1}}$ and $t_{1-d} \leftarrow r$ for both choices of $d \in \{0,1\}$. The simulation provided by $\mathcal{B}$ is perfect and reveals nothing to $\mathcal{A}$ about d unless the chosen random values for the indeterminates give rise to a nontrivial equality relation (identical polynomial in any of the lists L_1, L_T) between the simulated group elements that was not revealed to $\mathcal{A}$, i.e., when we assign $A \leftarrow \alpha, \Gamma \leftarrow \gamma$, and either $T_0 \leftarrow \frac{z\gamma}{\alpha^{n+1}}, T_1 \leftarrow r$ or the converse $T_0 \leftarrow r, T_1 \leftarrow \frac{z\gamma}{\alpha^{n+1}}$. This happens only if for some i, j one of the following holds:

- $F_{1,i}(\alpha, .., \alpha^n, \alpha^{n+2}, \ldots, \alpha^{2n}, \gamma, z, \frac{\gamma}{\alpha^{n+1}}, r) - F_{1,j}(\alpha, .., \alpha^n, \alpha^{n+2}, \ldots, \alpha^{2n}, \gamma, z, \frac{\gamma}{\alpha^{n+1}}, r) = 0$, yet $F_{1,i} \neq F_{1,j}$,
- $F_{T,i}(\alpha, .., \alpha^n, \alpha^{n+2}, \ldots, \alpha^{2n}, \gamma, z, \frac{\gamma}{\alpha^{n+1}}, r) - F_{T,j}(\alpha, .., \alpha^n, \alpha^{n+2}, \ldots, \alpha^{2n}, \gamma, z, \frac{\gamma}{\alpha^{n+1}}, r) = 0$, yet $F_{T,i} \neq F_{T,j}$,
- any relation similar to the above in which $\frac{\gamma}{\alpha^{n+1}}$ and r have been exchanged.

We now determine the probability of a random occurrence of a non-trivial numeric cancellation. Since $F_{1,i} - F_{1,j}$ for fixed i and j is a polynomial of degree at most $2n$, it vanishes for random assignment of the indeterminates in Z_p with probability at most $\frac{2n}{p}$. Similarly, for fixed i and j, the second case occurs with probability $\leq \frac{4n}{p}$. The same probabilities are found in the analogous cases where $\frac{\gamma}{\alpha^{n+1}}$ and r have been exchanged.

Now, absent of any of the above events, the distribution of the bit d in $\mathcal{A}$'s view is independent, and $\mathcal{A}$'s probability of making a correct guess is exactly $\frac{1}{2}$. Thus, by summing over all valid pairs i, j in each case, we find that $\mathcal{A}$ makes a correct guess with advantage $\leq 2(\binom{\tau_1}{2}\frac{2n}{p} + \binom{\tau_T}{2}\frac{4n}{p})$. Since $\tau_1 + \tau_T \leq q + 2n + 2$, we have $\epsilon \leq \frac{16n(q+n+1)^2}{p}$, as required.

A.3 Proof of Theorem 4

We will show trapdoor privacy that is the challenge keyword is indistinguishable from the same length random keyword. To show this we will present two games $\mathcal{G}_1, \mathcal{G}_2$. In $\mathcal{G}_1$ (*ideal* game), $\mathcal{SIM}$ chooses uniformly random r_1, r_2 values while in $\mathcal{G}_2$ (*real* game), $\mathcal{SIM}$ follows the protocol $r'' = r_1 + r_2$ and we show that these two games are indistinguishable.

Proof. We first consider $\mathcal{G}_1$. Suppose there exists an adversary $\mathcal{A}$ that distinguishes between challenge keyword w^* from random keyword with advantage ϵ. Then we construct a simulator $\mathcal{SIM}$ that wins the n-DDHI game as follows. Once $\mathcal{SIM}$ gets a n-DDHI instance $g, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}, g^\gamma, h, Z = h^{\frac{\gamma}{\alpha^{n+1}}}$, the game between $\mathcal{SIM}$ and $\mathcal{A}$ is as follows:

- **Init**: $\mathcal{A}$ gives $\mathcal{SIM}$ the challenge keyword w^*, user index i^* that he wants to be challenged on.

- **Setup**: $\mathcal{SIM}$ lets $g, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}$ in the simulation be as in the instance and picks $a_1, b_1, a_2, b_2, \zeta, \zeta', \theta, \theta'$ values from Z_p. He also sets $h_{0,1} \leftarrow g^{-w^*\zeta+\zeta'}$, $h_{1,1} \leftarrow g^{\zeta}$, $h_{0,2} \leftarrow g^{-w^*\theta+\theta'}$, $h_{1,2} \leftarrow g^{\theta}$, He computes public key parameters as $h_{0,1}^{a_1} = (g^{a_1})^{-w^*\zeta+\zeta'}$, $h_{1,1}^{a_1} = (g^{a_1})^{\zeta}$, $h_{0,1}^{b_1} = (g^{b_1})^{-w^*\zeta+\zeta'}$, $h_{1,1}^{b_1} = (g^{b_1})^{\zeta}$, $h_{0,2}^{a_2} = (g^{a_2})^{-w^*\theta+\theta'}$, $h_{1,2}^{a_2} = (g^{a_2})^{\theta}$, $h_{0,2}^{b_2} = (g^{b_2})^{-w^*\theta+\theta'}$, $h_{1,2}^{b_2} = (g^{b_2})^{\theta}$, $v = g^{\gamma}$ and gives them to both $\mathcal{A}$ and $\mathcal{S}_{aid}$. $\mathcal{SIM}$ generates user secret keys sk_i by running Setup algorithm. As a note that, the simulator does not know the values α, γ. Then, $\mathcal{SIM}$ gives public parameters to $\mathcal{A}$ and $\mathcal{S}_{aid}$.
- **Query:** $\mathcal{A}$ makes the following queries to $\mathcal{SIM}$ adaptively. For encryption query (S_ℓ, w_ℓ),
  If $w_\ell \neq w^* \wedge i^* \notin S_\ell$, the simulator picks k, k_1, k_2 encryption keys for file ℓ and gives the computed values,
  $hdr_{\ell,1} = g^{a_1(w_\ell - w^*)\zeta k_1} g^{a_1\zeta' k_1}$, $hdr_{\ell,2} = g^{(w_\ell - w^*)b_1\zeta(k-k_1)} g^{b_1\zeta'(k-k_1)}$, $hdr_{\ell,3} = g^{a_2(w_\ell - w^*)\theta k_2} g^{a_2\theta' k_2}$, $hdr_{\ell,4} = g^{(w_\ell - w^*)b_2\theta(k-k_2)} g^{b_2\theta'(k-k_1)}$, $hdr_{\ell,5} = g^k$, $hdr_{\ell,6} = \left(g^{\gamma} \prod_{j \in S_\ell} g_{n+1-j}\right)^k$, $hdr_{\ell,7} = e(g_{n+1}, g)^k, S_\ell$ and gives them to $\mathcal{A}$ and gives $hdr_{\ell,5}, hdr_{\ell,6}, hdr_{\ell,7}, S_\ell$ to $\mathcal{S}_{aid}$.
  if $i^* \in S_\ell \vee w^* = w_l$: The simulator outputs $\perp$.
- **Guess:** $\mathcal{SIM}$ assigns h is the form of $h = g_{n+1+i}^{r''}$ (r'' is unknown to $\mathcal{SIM}$). He picks random $\rho_{i1}, \rho'_{i1}, \rho_{i2}, \rho'_{i2}, r, r'$ then computes the trapdoor as $tr_1 = Z g^{a_1 b_1 \zeta' X} g^{a_2 b_2 \theta' Y}$, $tr_2 = g^{a_1 X}$, $tr_3 = g^{a_2 Y}$, $tr_4 = g^{b_1 X}$, $tr_5 = g^{b_2 Y}$, $tr_6 = h = g_{n+1+i}^{r''}$, where $X = \rho_{i1} r + \rho'_{i1} r'$, $Y = \rho_{i2} r + \rho'_{i2} r'$. Then, he gives $tr_1, tr_2, tr_3, tr_4, tr_5, tr_6, r_2$ to $\mathcal{A}$ and r_1 to $\mathcal{S}_{aid}$.
- **Output**: $\mathcal{A}$ outputs a bit b. To conclude, the simulator forwards b as its own answer in the n-DDHI game. If the n-DDHI instances are well formed, the adversary outputs $b = 0$ which is a random keyword, otherwise it outputs $b = 1$ which keyword is w^*.

Analysis: Under the restriction in the encryption phase, $\mathcal{S}_{aid}$ does not store keyword ciphertext that is formed by the challenge index (the public key of the challenge user g_{i^*}). Therefore, the challenge trapdoor and any keyword ciphertexts are not going to be compatible when $\mathcal{S}_{aid}$ computes function of r_1, C'. It means that the challenge trapdoor is independent of w^*, the adversary's best success probability is $1/2$ when $\mathcal{A}$ outputs $b = 0$ if the game is totally random. The success probability is $\Pr[\text{win}_{\mathcal{A}}(\lambda)] < 1/2 + \epsilon$ when $\mathcal{A}$ outputs $b = 1$ if the n-DDHI instances are well formed. So, $\mathcal{SIM}$ breaks n-DDHI assumption with probability $|\Pr(\mathcal{A}(b = 1) = 1) - \Pr(\mathcal{A}(b = 0) = 1| = 1/2 + \epsilon - 1/2 = \epsilon$, which is non-negligible. So

$$\left|\Pr\left[\mathcal{G}_1^{\mathcal{A}}\right]\right| \leq \epsilon$$

In $\mathcal{G}_2$, $\mathcal{SIM}$ follows real game, chooses r_1 and r_2 such that $r'' = r_1 + r_2$ and gives r_1 to $\mathcal{S}_{aid}$ and r_2 to $\mathcal{A}$. In the real game, $\mathcal{A}$ can not ask encryption queries that user index $i^* \in S_\ell$ or $w^* = w_l$. This results $\mathcal{A}$ is not able to test if the keyword is w^* or a random keyword. We argue that since $\mathcal{A}$ gets the function of r_1 and C' and he is not able to learn any non-trivial information about r_1 under the restriction of the game. The function of r_1 and C' is totally

random to $\mathcal{A}$ for every r_1 since for each encryption of a keyword w the challenger chooses fresh (random) elements (k, k_1, k_2) from Z_p. Basically, the information (randomized ciphertext) given to $\mathcal{A}$ is semantically secure. Let ε is the advantage of $\mathcal{A}$ winning the semantic security encryption then we can say $\mathcal{SIM}$ breaks n-DDHI assumption with probability $|\Pr(\mathcal{A}(b=1)=1) - \Pr(\mathcal{A}(b=0)=1| = 1/2 + \epsilon - (1/2+\varepsilon) = \epsilon - \varepsilon$. Then,

$$\left|\Pr\left[\mathcal{G}_2^{\mathcal{A}}\right]\right| \leq \epsilon - \varepsilon$$

As a result,

$$\left|\Pr\left[\mathcal{G}_1^{\mathcal{A}}\right]\right| - \left|\Pr\left[\mathcal{G}_2^{\mathcal{A}}\right]\right| \leq \varepsilon$$

This completes the proof.

□